Intro to Machine Learning
Report for Code Assignment 1

Tal Grossman	201512282
Din Carmon	209325026
Amir Sharif Jamal	213850811

Tal Grossman 201512282, Din Carmon 209325026, Amir Sharif Jamal 213850811

How-to-run and implementation details

How to run

- Download all Python files to a single directory. (for example “code_assignment_1”)
- Run the “main.py” (for example python main.py)
    - This will run both parts and save the results
- If you wish to change hard coded hyperparameters look for Upper Case constants in each part.

Implementation details:

Dataset handling and retrieval:

- Dataset: tabular breast cancer classification from:
    - https://scikitlearn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html#sklearn.datasets.load_breast_cancer
- Implemented in “get_data.py” under the “get_dataset” function.
- shuffled randomly with a hyperparameter random seed.
- Used 80% for training and 20% for testing.

Part 1:

- Implemented in part1_neural_network.py as well as required utility functions.
- Runs the training and computes all results including train and test loss over epochs, and final test accuracy per experiment (see details in part 1 section).
- Plot and save all experiments results including best final weights.

Part 2:

- Implemented in part2_decision_trees.py
- Runs on the training data and computes a decision tree using the ID3 algorithm.
- Save the decision tree.

Tal Grossman 201512282, Din Carmon 209325026, Amir Sharif Jamal 213850811

Part 1: Single-Layer Neural Network with Gradient Descent

This section will include the implementation details and results of a single-layer neural network.

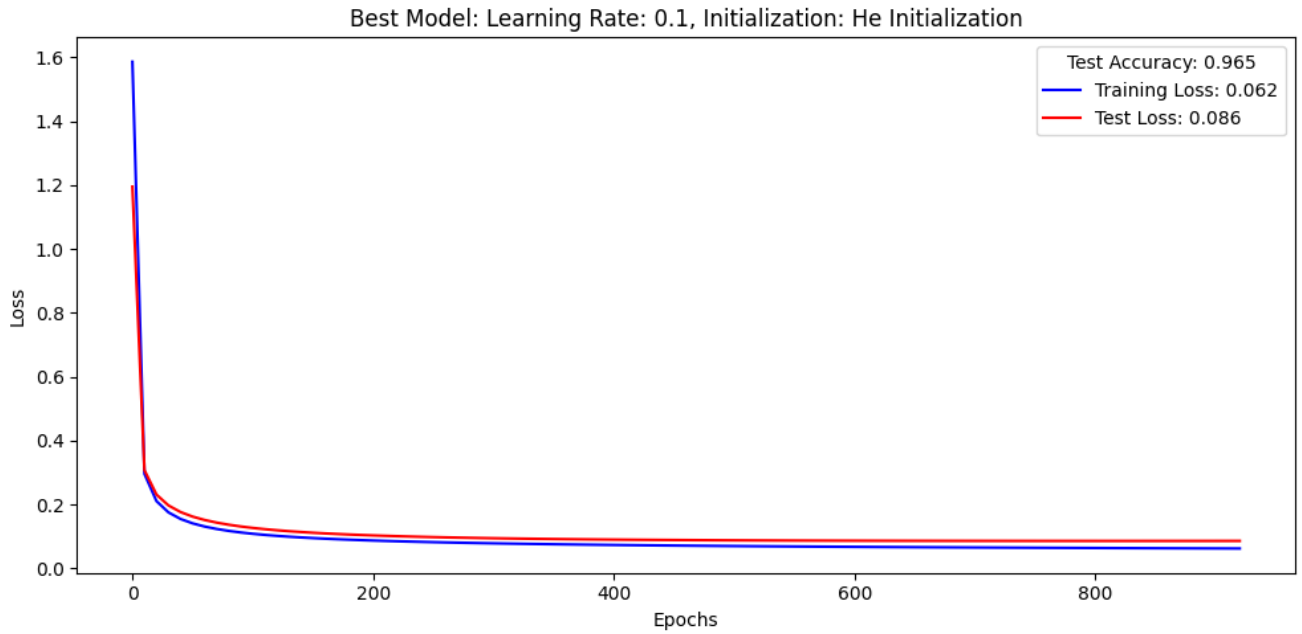

Figure 1: The best experiment training and testing losses plot with a test accuracy of 96.5%. Experiment configuration: learning rate of 0.1 and He Initialization.

Methodology:

Training:

- **Model**:
  A single-layer neural network with a sigmoid activation and cross-entropy loss functions.
- **weights initialization**: we used both He initialization and Normal Initialization.
- About the He initialization:
  He initialization is a weight initialization strategy designed to improve training stability and prevent vanishing or exploding gradients, especially in deep networks. It carefully scales the initial weights based on the number of input connections to each neuron, ensuring that the activations neither become too small (hampering learning) nor too large (causing saturation). This leads to faster and more reliable convergence during training.
  - Formula: $W \sim N(0, \sigma^2)$ where $\sigma^2 = 2 / n$
    Where
    - W represents the weight matrix.
    - $N(0, \sigma^2)$ denotes a Gaussian distribution with mean 0 and variance σ^2.
    - n is the number of features in our case.
  - References:
    - https://arxiv.org/abs/1502.01852

        - https://medium.com/@shauryagoel/kaiming-he-initialization-a8d9ed0b5899
- **Stopping Conditions:**
    - We tried 2 stopping conditions
        - Hyperparameter of maximum iterations (epochs) set to 1000.
            - This was the stopping condition we used for the results given in this report.
        - Model Improvement “LEARNING_PATIENCE”.
            - This stooping condition should stop the training process if the validation loss (the test loss in our case) did not improve by more than some factor for some few last iterations.
                - Look for STOPPING_CONDITION / LEARNING_STOP_CRITERIA_NUM_OF_LAST_ROUNDS in code to change the stopping condition.
    - Both stopping conditions reached a convergence and showed good results. However, the second condition assumes reaching a convergence which is not necessarily true using a gradient descent approach. Therefore, we chose the first stopping condition.

Results:

Learning rates	Initializations
1. 0.1 2. 0.01 3. 0.001	1. He initialization, References: 2. normal initialization

Table 1: All experiment configurations

We conducted **all combinations** of the configurations shown in table 1.
Since the gradient decent approach is a heuristic iteration optimization algorithm the learning rates play a crucial role in the convergence rate and in achieving local or global minimum. A small learning rate can cause a slow convergence or none at all. A too big learning rate could cause oscillations in loss convergence and may end up in a non-optimized model.
Furthermore, in GD algorithms, when the dataset is limited in size, as we had in this assignment, the weights initializations can have a great effect on the convergence rate and the final model fine-tuned weights.

Tal Grossman 201512282, Din Carmon 209325026, Amir Sharif Jamal 213850811

Results and Analysis

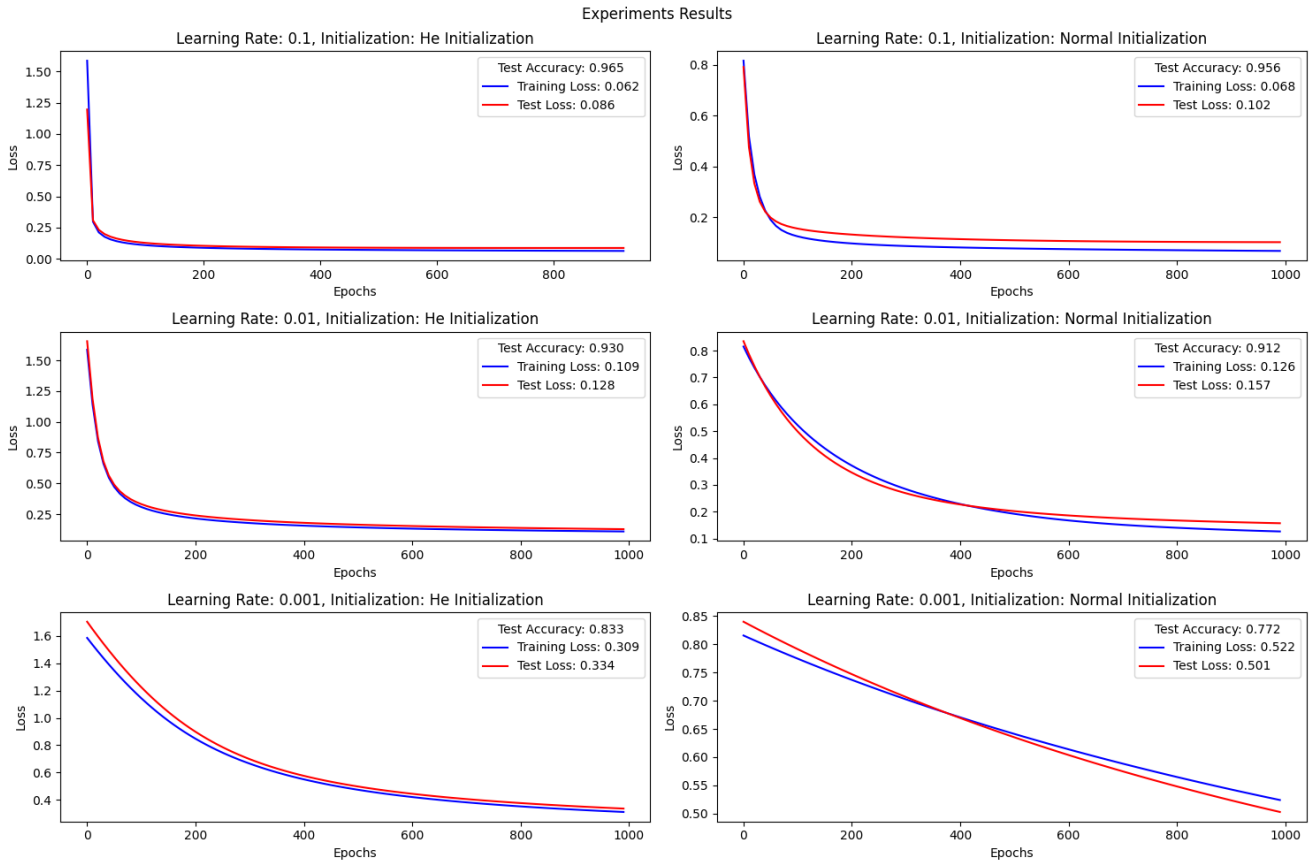

Figure 2: All experiment training and testing losses plots – top to bottom: learning rates 0.1, 0.01, 0.001. Left He initialization. Right: normal initialization.

- As expected, A too-small learning rate was too slow to converge. With a relatively high learning rate of 0.1 we got fast convergence, and it was also “stable” to not cause loss oscillations.
- The He Initialization did improve the convergence rate by having stable gradients.
- The best result, seen in Figure 1 and at the top left of Figure 2, is with a learning rate 0.1 and He initialization. It was best in terms of test accuracy and lowest test loss.
- <u>Strengths and weaknesses:</u>
    - The single layer with gradient descent model is a strong model because it reaches good accuracy fast (For the given data). It is also simple and light on hardware consumption, especially with tabular small feature datasets. But it has its weaknesses, the first being it highly depends on the hyperparameters as seen in Figure 2, and the dataset labels, size, and its features.
    - A one-layer neural network can only represent an activation result of a linear transformation of the input. This means it cannot capture complex patterns in the data.

Tal Grossman 201512282, Din Carmon 209325026, Amir Sharif Jamal 213850811

Model’s weights and biases:

Model’s learnable parameter	Shape	output
W	(num_features, 1) = (30, 1)	*[-0.12239466538368492, -0.6240584065446445, -0.3091616895215014, -0.040622776737171455, -0.18613240116829266, -0.22017592775013775, -0.4273080974421825, -0.8121965613751213, -0.41000495485251026, 0.3490237968403959, -1.0140605579748623, 0.046170652769180656, -0.6333119397322726, -0.7779752846083727, 0.09644302636240362, 0.6509392924154866, 0.40084682378541114, -0.14860160216450363, 0.16397575898746916, 0.4021082892572181, -1.5204364651363507, -0.9275590409806291, -0.5701598777239533, -1.032247065528047, -0.4098857845159833, -0.5593456616844367, -0.7608466363385203, -0.8979880253122238, -0.3697128047526266, 0.05327887273338164]*
b	(1, 1)	*[0.37718557531896707]*

Table 2: Weights and bias for the best configuration.

As requested, we attach in Table 2 the final trained **best** weights and bias. Run code and get the rest.

Tal Grossman 201512282, Din Carmon 209325026, Amir Sharif Jamal 213850811

Part 2: ID3 Algortihm

We used the ID3 algorithm to create a decision tree with the same training data used at part 1.
The implementation includes a split class which is used to represent the conditon in an inner node of the tree. It can handle both categorical and numerical data.
The final tree:

```
x[7] < 0.07482458016741707 ?
 No: x[22] < 0.2185919341055576 ?
   No: 0
   Yes: x[21] < -0.032116573421551185 ?
     No: x[12] < -0.6683206434470212 ?
       No: 0
       Yes: x[0] < -0.26336401810609117 ?
         No: 1
         Yes: 0
     Yes: x[27] < 0.6500809502123001 ?
       No: 0
       Yes: 1
 Yes: x[20] < 0.11406316838261259 ?
   No: x[1] < -0.833008236294527 ?
     No: x[17] < -0.3036787300787817 ?
       No: 1
       Yes: 0
     Yes: 1
   Yes: x[10] < 0.7417935140239024 ?
     No: x[4] < -0.6291225182251122 ?
       No: 0
       Yes: 1
     Yes: x[21] < 0.7234786189206999 ?
       No: x[20] < -0.3912182048164362 ?
         No: x[0] < -0.19236046906767404 ?
           No: 1
           Yes: x[1] < 1.7872693480991924 ?
             No: 1
             Yes: 0
         Yes: 1
       Yes: 1
```

Tal Grossman 201512282, Din Carmon 209325026, Amir Sharif Jamal 213850811

Part 3: Comparison

- Comparing the accuracy of the best configuration of the NN, to the ID3 built decision tree:

```
/Users/dinc/miniconda3/envs/Assignment1/bin/python /Users/dinc/Documents/projects/StatisticalMachineLearningCodeAssignments/Assignment1/main.py
Graphs + Best 1 layer NN configuration were saved to: ./results/part1
Best configuration 1 layer NN accuracy: 0.9912280701754386

Decision tree was saved to: ./results/part2/decision_tree.txt
Accuracy of ID3 Constructed decision tree: 0.9385964912280702

Process finished with exit code 0
```

- The NN approach yields in a better accuracy.
- Based on the foundlings:
- The NN is superior to the decision tree approach for a better accuracy.
- The decision tree yields a more informative description of the prediction algorithm and may help in identifying inner correlations / understanding of the data for future modeling / conclusions on the data. In other words, With the NN, it is harder to understand why a specific decision was made compared to rule-based models like decision trees.
- Simpler structure of a 1 NN results in faster computation compared to deeper networks.
- The decision tree Can model complex, non-linear relationships between features and labels.
- ID3 trees can grow very deep, memorizing the training data instead of generalizing. Thus, are more proned to overfitting.
- ID3 are sensitive to small changes in the data, which can lead to a completely different tree structures.
- Decision trees splits are typically based on a single feature at a time, which might not capture interactions between features effectively.

<u>Discussion on overfitting / underfitting</u>

- A big gap between the train error and the test error is an indication for an overfitting. The NN results show the train and test losses are about the same, which means the NN did not overfit.
- However, the decision tree yields more than 90% accuracy. Therefore, although decision trees are more prone to overfitting, the overfit is not neccssarily drastic. But, perhaps modifying the ID3 algorithm by bounding the depth of the decision tree shall result in a better accuracy, which shall indicate an overfitting of the current ID3 algorithm.
- An underfit would be seen if the final train loss was high, which was not the case in either algorithms (The train loss for the ID3 algorithm is 0).